

IME : A TOOL FOR DEVELOPING KNOWLEDGE - BASED SYSTEMS

**A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY**

**By
R. G. BHANDI**

to the

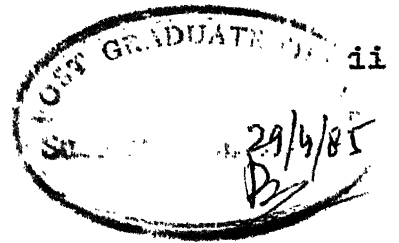
**INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
MAY, 1985**

12 JUN 1985

LIBRARY
CENTRAL LIBRARY

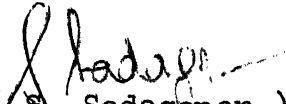
NO. A 82473

IMEP-1985-M-BHA-IME

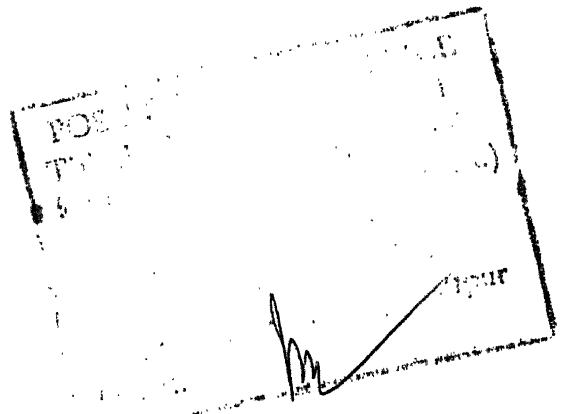


CERTIFICATE

This is to certify that the work entitled, IME: A TOOL FOR DEVELOPING KNOWLEDGE-BASED SYSTEMS, done by Shri Raghupati G. Bhandi has been carried out under my supervision and has not been submitted elsewhere for a degree.


(S. Sadagopan)
Assistant Professor
Industrial and Management Engg.
Indian Institute of Technology
Kampur 208 016

April, 1985



ACKNOWLEDGEMENTS

It is with immense pleasure and great respect that I express my deepest sense of gratitude to Dr. S. Sadagopan for his invaluable guidance and encouragement throughout my work.

I take this opportunity to express my sincere thanks to all members of IME family, for their constant inspiration.

I thank Swami Anand Chaitanya for his neat typing and Mr. Buddhi Ram Kandiyal for cyclostyling work.

R. G. Bhandi

CONTENTS

<u>Chapter</u>		<u>Page</u>
	ABSTRACT	v
I.	INTRODUCTION	1
	1.1 Artificial Intelligence	1
	1.2 Knowledge Based Systems	2
	1.3 Existing Expert Systems	4
	1.4 Advantages of Expert Systems	5
	1.5 Limitations of Expert Systems	6
	1.6 Motivation for the Current Work	6
II.	EXPERT SYSTEMS	9
	2.1 Introduction	9
	2.2 Fundamental Qualities of Expert Systems	9
	2.3 Components of an Expert System	10
	2.4 Tools for Building Expert Systems	14
III.	IME: SYSTEM DESIGN	16
	3.1 Introduction	16
	3.2 Knowledge Representation	16
	3.3 Inference Engine	22
	3.4 Explanation System	23
	3.5 Knowledge Base Editing	23
IV.	IME: SYSTEM ORGANIZATION	24
	4.1 Knowledge Representation	24
	4.2 Indexing	31
	4.3 Inference Engine	31
	4.4 Explanation System	39
	4.5 IME: Top-Level	41
V.	RESULTS AND CONCLUSIONS	42
	4.1 Summary	42
	4.2 Limitations	43
	REFERENCES	45
	APPENDICES	

ABSTRACT

IME is a computer program that can be used as a skeletal system for developing knowledge-based expert systems. IME has an inference engine that will become the heart of any expert system that is built on IME. Primary function of this subsystem is to achieve a given goal using knowledge about the domain. Tree searching techniques are used for this purpose. IME also has a convincing explanation system which produces explanations and justifications for the conclusions drawn and the questions asked.

We claim that such a skeletal system is very helpful in building expert systems because now the developer is relieved of designing and implementing the basic structure.

CHAPTER I

INTRODUCTION

1.1 ARTIFICIAL INTELLIGENCE:

Artificial Intelligence (AI) is that field of computer science where computers are made to perform tasks that ordinarily require human intelligence [4].

AI programming languages and tools are different from conventional programming languages in that the latter consist of two distinct parts: algorithm and data. Algorithm specify how to solve specific problems and data characterize parameters in the particular problem at hand. However, human beings seem to work in a totally different way. Human knowledge consists of elementary fragments of know-how and these fragments are applied properly for solving tasks. AI tools are nearer to human beings in this respect than the conventional programming tools in that they work in a way somewhat similar to the working of human beings.

The real power of AI programming tools comes from their ability to work with symbols, or the so-called symbolic manipulation. AI tools make no distinction between data and procedure. Depending on the context the same element may be treated as a data or as a procedure.

Intense research is going on in AI and has lead to several specialized fields. Current AI activities can be classified mainly into three fields:

1. Natural Language Processing
2. Robotics and Computer Vision
3. Knowledge Based Systems (KBS)

In the present work we are concerned only with KBS.

1.2 KNOWLEDGE BASED SYSTEMS:

AI has achieved considerable success in the development of KBS, since the activity started a couple of decades back. This area of AI has concentrated on the construction of high performance programs in specialized professional domains. The basis of KBS is the belief: human experts achieve outstanding performance because they are knowledgeable. If computer programs embody and use this knowledge, then they too should attain high levels of performance. This has proved to be true repeatedly in the short history of KBS. Systems have attained expert levels in several tasks. Success in this field has encouraged an emphasis on the knowledge that underlies human expertise and has simultaneously decreased the apparent significance of domain independent problem solving theory. It was pointed out that what really constitutes an expert is the knowledge he has in a particular domain rather than just high IQ. From this, a new set of principles, tools, and

techniques has emerged, which forms the basis of Knowledge Engineering (KE) [1].

Knowledge in any speciality is usually of two sorts: public and private. Public knowledge includes the published definitions, facts, and theories. But expertise is usually more than just the public knowledge. Human experts generally possess private knowledge that consists largely of rules of thumb that have come to be called as heuristics. These heuristics enable human expert to make educated guesses when necessary, to recognize promising approaches and to deal effectively with errorful or incomplete data.

Knowledge Based Systems or Expert Systems - as they are called interchangeably - differ in important ways from both conventional data processing systems and systems developed in other branches of AI. In contrast to traditional data processing systems AI applications generally involve several distinguishing features which include symbolic representation, symbolic inference and heuristic search. A simple AI task often yields to one of the formal methods developed for the above features. But expert systems differ from the broad class of AI in several respects.

1. Expert systems operate at expert levels of performance.
2. Emphasis is on domain-specific problem solving strategies rather than general problem solving techniques.

3. They employ self knowledge to provide explanations or justifications for the conclusion they have drawn.

1.3 EXISTING EXPERT SYSTEMS:

Several expert systems are developed which now operate at the levels of human experts. Some of them are listed below [5]:

1. MYCIN incorporated over 400 heuristic rules to diagnose and treat infectious diseases. It has a powerful explanation system that explains the diagnosis or reasons for asking certain questions.
2. HEARSAY-II has thousand word vocabulary and understands speech. A global data base called black-board through which several independent, cooperating expert systems communicate constitutes the "brain" of this speech understanding system.
3. CADUCEUS embodies more knowledge of internal medicine than any human being has, and can correctly diagnose complex test cases that stymie human experts.
4. PROSPECTOR is a mineral expert that helps in discovering mineral deposits. It has been reported that PROSPECTOR discovered Molybdenum deposits whose ultimate value may be ten billion dollars.
5. DENDRAL is an expert in chemical analysis and is helping hundreds of international users daily.

1.4 ADVANTAGES OF EXPERT SYSTEMS:

1. Human-like Processing: The way in which information is processed by most types of expert systems is very close to that of human beings. They operate at the level of rules of thumb and in terms of concepts and relations, rather than the steps of a procedure in conventional programming. Moreover most of the expert systems provide a human window so that the user can get a feel about the proceedings of the problem. He also can interrupt and guide the proceedings if he so desires.

2. Ease of Expression: The language in which knowledge bases are expressed is close to the natural language with which human beings are familiar. Mostly if-then rules are used for coding knowledge. This is the most common way in which human experts express their knowledge as well.

3. Flexibility: Expert systems are able to embody rules of thumb that experts tend to carry in their head as well as the more formal knowledge like the laws of physics, chemistry etc. Also in some cases experts will not be knowing what all they carry in their head. A knowledge engineer-who is an expert in extracting such knowledge from experts - may bring to light several of the things which an expert uses but of which he is not aware of. This results in a better understanding of knowledge. Also this provides a good alternative

to the conventional way of storing knowledge in books. This may prove to be a better method of knowledge transfer than that through books.

4. Uncertainty: Expert systems can take care of uncertainties or even contradictions - which are the characteristics of most of the real life problems. This makes them all the more relevant to real life.

1.5 LIMITATIONS OF EXPERT SYSTEMS:

Today's expert systems have the following shortcomings:

1. They are unable to recognize or deal with problems for which their own knowledge is inapplicable or insufficient.
2. They have no independent means of checking whether their conclusions are reasonable.
3. Explications of their reasoning processes is frequently silent on fundamental issues.

In other words, today's expert systems fall well short of on dimensions requiring general intelligent behaviour. They are more akin to idiot savants than real human experts [6].

1.6 MOTIVATION FOR THE CURRENT WORK:

It was mentioned earlier that highly successful applications of AI in the field of KBS exist. Their accomplishments indicate that the field of Expert Systems is maturing rapidly.

However, the scientific and technical bases that support this field have achieved only limited development [1]. Each new application requires creative and challenging work, although some principles and systemizations have emerged. At this point expert systems field is highly experimental one with little in the way of general theory.

Expert systems are developed for varied applications - from mineral exploration to medical diagnosis, from Math assistant to Chess. Conditions prevailing in each applications are characteristics of a very narrow class of problems. Knowledge content may vary from "deep narrow" to "wide shallow". But still all of them have something in common. They are:

1. Expert systems have to have some kind of knowledge representation.
2. An inference engine to deduce things using knowledge and evidences.
3. An explanation system that justifies and explains the conclusions drawn and the questions asked.
4. User interface or dialogue system.

It is logical to think of developing some tools to accomplish the above. This thesis work is a step in this direction.

This general tool may have a knowledge representation method that would hold diverse knowledge, an inference engine

that would achieve goals based on the knowledge, a general mechanism of deduction (or inference), and an explanation system that readily adjusts itself to the vocabulary of any domain.

IME -a computer program that incorporates the above principles is discussed in this thesis. In Chapter II, we study expert-systems in general - mainly to grasp the common requirements, which can go into a general purpose tool. Chapter III deals with the system design while Chapter IV gives a detailed account of the actual implementation. In Chapter V we summarise the work with some comments.

CHAPTER II

EXPERT SYSTEMS

2.1 INTRODUCTION:

Expert systems are problem solving programs that solve difficult problems requiring expertise. They are termed knowledge-based because their performance depends critically on utilizing facts and heuristics used by experts.

2.2 FUNDAMENTAL QUALITIES OF EXPERT SYSTEMS:

It is difficult to define what constitutes an expert system. Rather than trying to define it, a discussion on the salient features of an expert system would be more meaningful at this stage.

Expert systems must have some expertise. The dictionary says that an expert is one who is skilled. However, this sense misses the mark within the context of expert systems. It takes more than good performance to make an expert system. For example, numerical analysis programs for solving differential equations perform well but fall short of the mark as AI expert systems. Knowing the well established principles, methods and laws do not qualify one as an expert. He should have something more than that.

Expert systems should possess general problem-solving ability in a domain. An expert system can be more or less intelligent, depending on the scope of its basic principles and the quality of its general-purpose reasoning processes.

Expert systems should also have robustness. As new, unanticipated patterns crop up, inflexible compiled solutions fail.

Features of an expert system can be summarized as follows [1]:

An expert system is one that has expert rules and avoids blind search, performs well, reasons by manipulating symbols, grasps fundamental domain principles, and has complete weaker reasoning methods to fall back on when expert rules fail and to use in producing explanations. It deals with difficult problems in a complex domain, can take representation appropriate for processing with its expert rules, and it can reason about its own knowledge (or lack thereof), especially to reconstruct inference paths rationally for explanation and self justification. An expert system works on one of these tasks: interpretation diagnosis, prediction, instruction, monitoring, planning and design.

2.3 COMPONENTS OF AN EXPERT SYSTEM:

An ideal expert system should have the following components.

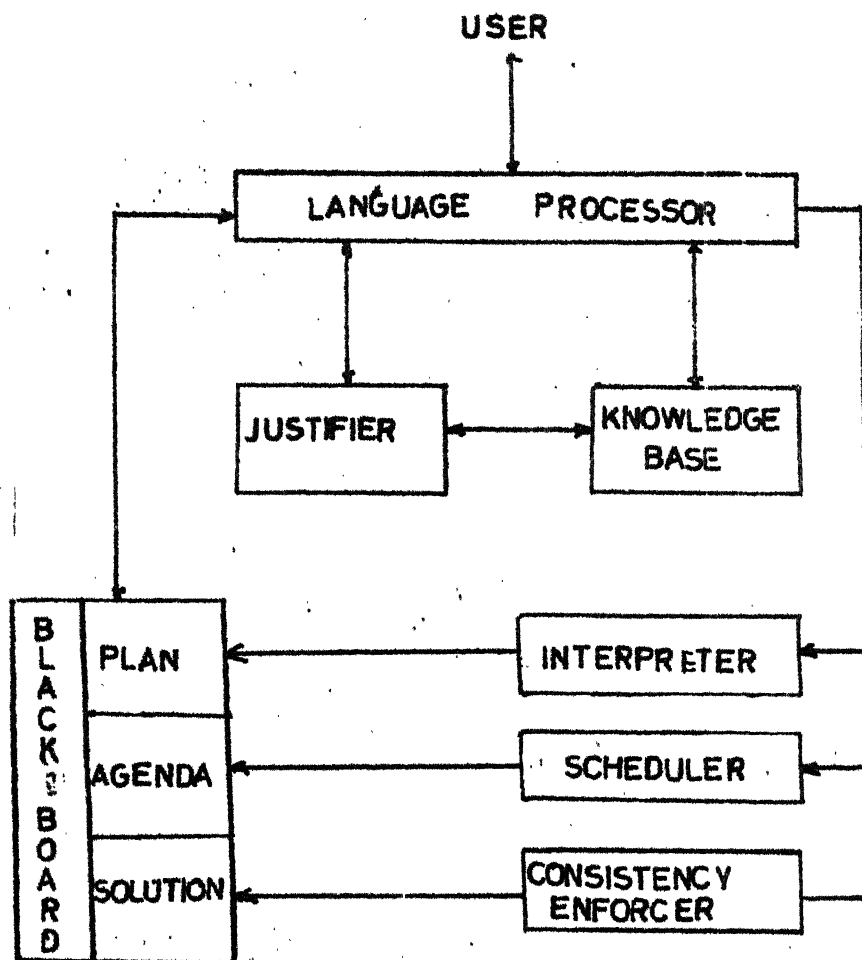


Fig. 2.1 Components of an ideal expert system

1. Language Processor:

User usually interacts with the Expert System in problem-oriented languages. User inputs are parsed into commands. Usually parsers are simple and exploit the domain specific characteristics of inputs. Key word recognition is sufficient in most of the cases. Conversely language processor formats (or does just slot-filling) the informations generated by the system including answers to questions, explanations and justifications for its behaviour and requests for data.

2. Knowledge Base (KB):

Knowledge Base records rules, facts and information about the current problem that may be useful in formulating a solution. Rules of the knowledge have procedural interpretation. They direct the inference engine towards achieving the set goal. Facts play only passive roles. They just say yes-no to the inference engine or at the most supply values to variables.

3. Black Board:

The concept of black board was first made use of explicitly in HEARSAY - a speech recognition system. However, every expert system uses some type of intermediate decision representation, but only a few use them explicitly. Three subdivisions are identified in a black board: plan, agenda

and solution. Plan elements describe the overall strategy for attacking the problem at hand. Agenda elements record the potential actions awaiting execution, which generally correspond to knowledge base rules that seem relevant to some decision placed on the black board previously. The solution elements represent the candidate hypothesis and decisions the system has generated thus far, along with the dependencies that relate decisions to one another.

4. Scheduler:

Scheduler decides the rule to be tried next. Scheduler, in some sense, judges the potential use of each of the alternatives maintained in agenda and selects the most promising one.

5. Interpreter:

Interpreter executes the agenda that is chosen by scheduler. Execution usually involves validating certain conditions finding particular bindings to certain variables and making appropriate changes in black board.

6. Consistency Enforcer:

Consistency Enforcer attempts to maintain a consistent representation of the solutions found. This usually involves implementing truth-maintenance procedures or adjusting the relative weightages of various views expressed or deduced about a single concept or entity.

2.4 TOOLS FOR BUILDING EXPERT SYSTEMS:

Attempts are made to develop tools or skeletons for building expert systems so that a particular system can be built in a short while without much programming efforts. Several of the successful expert systems were later generalized. For example MYCIN's success lead to EMYCIN which is a generalized version of MYCIN. Tools currently available are briefly discussed.

1. EMYCIN: EMYCIN basically a domain-independent version of MYCIN, is an appropriate skeletal system for developing a consultation program that can request data about a case and provide an interpretation or analysis. It is practically well suited to deductive problems.
2. KAS is the generalized version of PROSPECTOR. This is similar to EMYCIN but the inference engine differs slightly. Weightage is given to the data supplied by the user and some amount of forward chaining is done.
3. OPS5 incorporates general control and representation mechanisms. It provides the basic mechanisms needed for knowledge engineering. But it is not biased towards any particular problem-solving strategies or representational schemes. OPS5 allows the programmer to use symbols and represent relations between symbols, but no symbols or relations have predefined meanings. Meanings are entirely determined by the production rules given.

4. RLL is a relatively advanced system. It contains primitives and collection of tools for combining them. A large collection of types of slots, control mechanisms and inheritance schemes are available and the user makes his choice. RLL knows enough to combine them into what the user then perceives as a representation language.

5. ROSIE is basically a general purpose AI language. It can be used to build skeletal systems, but as such no skeletal system is given. This is useful when the problem at hand does not fit any of the ready skeletal schemes. Powerful facilities are given to develop an expert system.

CHAPTER III

IME: SYSTEM DESIGN

3.1 INTRODUCTION:

The aim of this thesis work is to develop a skeletal system using which expert systems can be readily built. This requires that domain specific aspects should be clearly separated from the domain independent ones and facilities should be provided to introduce the former into the system through rules or whatever knowledge representation chosen. Control structure of IME should be flexible so that it adapts itself to the diverse needs of different expert system domains. Key design problems are discussed in this chapter.

3.2 KNOWLEDGE REPRESENTATION:

Knowledge Based Systems are in a way simulation of human experts. Hence the knowledge representation method^{should} lend itself readily to hold the fragments of know-how given by an expert. At the same time it should be convenient for internal representation and manipulation.

Two methods of knowledge representation are discussed in literature.

3.2.1 Frames:

In simple terms frame can be viewed as a stereotypical

representation of any object concepts. It is typically represented as a data-structure whose "name" is that of the concept. It has various types of "links" to other frames. It also has slots each of which stands for an attribute of the object concept of interest. It can hold values of these attributes and procedures or demons that are invoked under certain conditions. From data structure point of view it is an extension of the name-attribute-value triples provided in LISP. For example knowledge about a concept called Table may be of the form shown below [7]:

name : TABLE

links: aTypeOf FURNITURE

; FURNITURE is another frame which is linked
to TABLE through aTypeOf.

slots: numberoflegs

(default 4)

(value)

; i.e. numberoflegs has a default
value of 4.

materialoftop

(default wood)

(value

(infer: if numberoflegs > 4 then marble
else wood))

; If the value of the attribute is asked
and if no value is stored, then the
procedure is invoked.

Frames offer the following advantages [7]:

1. They allow explicit representation of entities, attributes, relations, default values etc..
2. When certain parameters are not traced (no value is stored) they can be automatically traced by triggering the demons . Explicit representation of procedures for finding the values are not usually possible in more conventional data bases.
3. Frames can be related in a hierarchy. Attributes, values and triggers can be inherited from a frame that is higher in the hierarchy. This provides an easy way of defining a class of data items that share attributes procedures and default values.

Frames have the following limitations. Frames are powerful only when hierarchies exist. Otherwise overheads on manipulation of complex data structure go as waste. Also price paid for hierarchy is modularity. Additions or deletions to the knowledge base are to be made carefully considering the links between frames. Expert systems, especially in the beginning stages of their service need extensive revision of knowledge base, and so modularity may be an important requirement.

Use of frames for knowledge representation has been limited to a handful of application, e.g. MDX and PSN.

3.2.2 Production Rules:

Production rules are of the form IF <antecedent> THEN <consequent>. Both antecedent and consequent describe states of the production system. Theoretically this is slightly different from IF <premise> THEN <action> in that the latter is primarily meant for triggering some action while the former is just to change the picture of the system through logical conclusions. However, in usage they are not different either in representation or execution. Following is an example of a production rule in MYCIN [5].

Rule 050

```
IF      1) The infection is primary bacteremia,
        2) The site of culture is one of sterile sites, and
        3) The suspected portal of entry of the organism is
           the gastro intestinal tract
THEN    there is suggestive evidence (.7) that the identity of
        the organism is bacteroids.
```

This rule has an internal representation in LISP as:

```
PREMISE  (SAND (SAME CNTXT INFECT PRIMARY-BACTEREMIA)
           (MEMBF CNTXT SITE STERILESITES)
           (SAME CNTXT PORTAL GI))
ACTION   (CONCLUDE CNTXT IDENT BACTEROIDES TALLY .7)
```

Production Rules are widely used for knowledge representation because of the following advantages they offer.

1. General task of deduction is one that fits quite well into the situation/action character of production rules. There is therefore less transformation necessary between the knowledge as expressed by the expert and its final encoding inside the system.
2. Rules are by themselves comprehensible "chunks" of knowledge, since they carry in their premise a full specification of their applicability. Their independence also facilitates the incremental growth of the knowledge base. Rules can be added one by one and performance improves with each addition.
3. Rules are retrieved and invoked on the basis of the contents of their antecedent (rather than just their name) and there is no reference between two rules. As a result adding and changing a rule is a far easier task ~~than~~ would be the case if there were extensive reference to the rule s names in many places.
4. Rules seem to capture a chunk of knowledge of appropriate size.
5. Rules are for the most part, what may be labeled a single level mechanism as far as the user is concerned. They are composed of elements that are conceptual primitives and require no further decomposition to be understood.

Apart from these implementational advantages productions rules have cognitive relevance to human thinking. Young [3,pp.37-38] argues that production system is a very close model of human cognition. He gives the following psychological phenomena in system model.

1. Stimulus Dependence and Independence:

Human Cognitive behaviour varies in the extent to which it is under the control of or independent of information in the immediate external task environment. Because of its data-driven character, a PS is well able to represent behaviour at any point on the scale from data-dependencies to independence. In production system a rule is brought into action when its left-hand side matches with the current state. Thus each rule can be thought of as a demon constantly on the watch for its evoking conditions.

2. Knowledge Based Cognition:

The kinds of human activity closest to the spirit of expert system are those cases in which the choice between a number of different possible ways of proceeding is made on the basis of a vast store of task-specific knowledge accumulated by the subject over a period of years.

3. Additivity and independence of production rules leads to their potential for displaying additivity, i.e. the possibility of enlarging the production system simply by adding to it more rules. This property in turn suggests that production systems

may be highly suitable for modelling the phenomena of human cognitive development since it should be possible to represent a child's growing ability as a gradually expanding production system.

These aspects have made production rules the most widely used representation of knowledge in expert systems.

It may be observed that characteristics of frames or atleast the essence of frames can be captured in production rules also with some extra rules and allowing special functions (to be discussed in Chapter IV). But it is not possible the other way round. Hence we selected production rules for representing knowledge in IME.

3.3 INFERENCE ENGINE:

Following assumptions were made about the domains for which IME can readily produce an expert system.

1. It is possible that there may be more than one realization of any relation. In otherwords predicates (or relations) may have more than one set of values in the data-base.
2. Rules are such that recursive calls to the same rules are not ruled out and if data-base fails to provide values to the functions concerned, search may go infinitely deep.
3. Knowledge representation requires the use of some special function i.e. functions with pre-assigned meaning and thus correspond to some procedure.

Above three aspects cover a wide range of domains and most of them become degenerated case of the above. On these considerations IME's inference engine was designed to take care of the above complexities. At the same time switches are provided to work efficiently in degenerate cases also without the extra burden of book keeping to take care of other complexities.

3.4 EXPLANATION SYSTEM:

Explanation should serve two purposes. First it should give justifications and explanations to the end user of the expert system that is developed using IME. Second it should aid the developer in tracing the inference mechanism for debugging. For end user explanation has to be in a form that makes sense to him which means it has to be in English rather than the internal representation. For debugging in the development stage what is important is the details of inference path that can be revealed IME's explanation system is designed on these grounds.

3.5 KNOWLEDGE-BASE EDITING:

Rules need modification in development stage. A structure based editor for knowledge-base is desirable.

CHAPTER IV

IME: SYSTEM ORGANIZATION

Overall structure of the system is shown in Fig. 4.1. User interaction is through interrogator or explanation system. Interrogator is a command recognition system that would trigger appropriate action. Knowledge Base Manager is meant for organizing the knowledge base consisting of rules facts and dynamic data and fetching relevant knowledge chunks on request from other systems. Inference engine is the heart of the system which carries out the deduction. Explanation system directly interacts with the user and supplies the justification and explanation for the result arrived at by the inference engine or about any question that is thrown at the user. When inference is in progress, any useful piece of information that was deduced are stored in Dynamic Data Base.

4.1 KNOWLEDGE REPRESENTATION:

Knowledge is coded as production rules in IME. Internal representation is a list like the one given below.

```
((SISTER-IN-LAW ?X ?Y) <= (Brother ?X ?Z) (WIFE ?Z ?Y)))
```

However with the translation facility this can be printed as

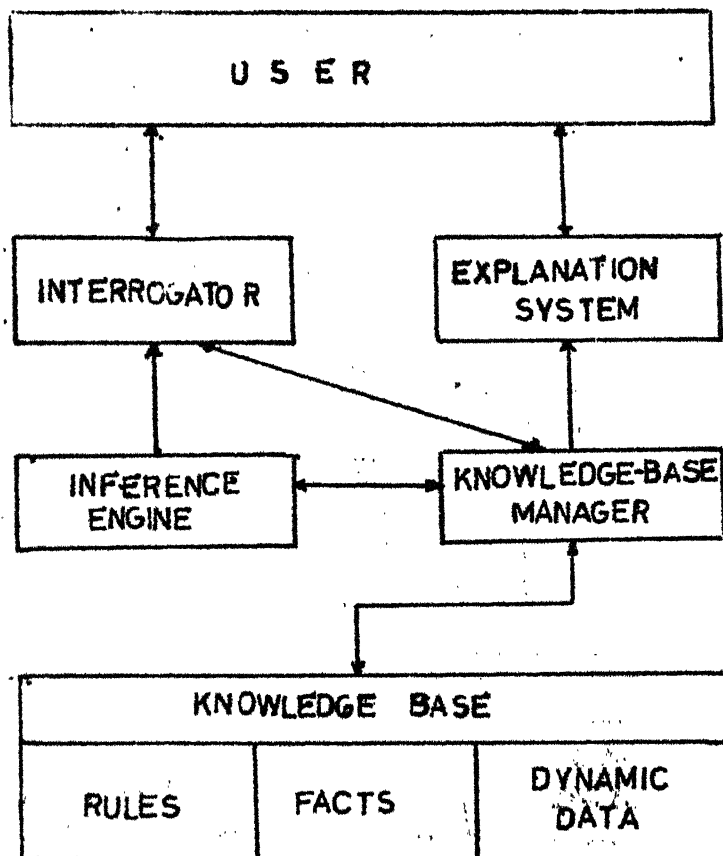


Fig. 4.1 IME System Organization

IF

1. Brother of X is Z, and
2. Wife of Z is Y

THEN it can be concluded that

Y is sister-in-law of X

BNF of a rule is given below:

```

<Rule>      ::= <consequent> <= <antecedent>
<consequent> ::= <function>
<function>   ::= (<function-name>{<argument>})
<function-name> ::= <identifier>
<argument>    ::= <identifier>/<predicate-variable>
               /<special-variable>
<antecedent> ::= ({<clause>})
<clause>     ::= <function>/(<NOT <function>>)/<special-function>

```

A function is anything of the form (function-name argument argument ...). IME attaches no meaning to a function. It is left to the user. However, special functions are the ones which are pre-defined and will have a LISP procedure in their name. Examples are EQUAL, ONE-OF, GREATER-THAN

At the time of development certain details about the functions used are to be supplied to IME to give IME a good interactive capability.

1. TRANS: This property is required to produce a translation of a function from internal list representation to English sentence.

For example SISTER-IN-LAW may have a TRANS property as

(~~#~~ 2 is sister-in-law of ~~#~~ 1)

IME understands that any occurrence of ~~#~~ N has to be replaced by N-th argument of the function before printing. Thus

(SISTER-IN-LAW LAXMANA SEETHA)

will be translated as

SEETHA is sister-in-law of LAXMANA.

2. ASKABLE: This is to tell IME whether facts about the function can be asked. This is valid only for functions with only one argument which can be supplied by the user.

3. PROMPT: This is the message that has to be printed as a prompt to get facts about the function from the user. For example AGE may be given a PROMPT property of

Please tell me your age.

Special Functions:

These are the functions for which LISP function definitions are available. This was thought to be necessary for doing some comparison of values or even some number crunching. For example (GREATER-THAN ?AGE 50) is a special function. Any predicate variables like ?AGE (to be explained shortly) are replaced by their bindings if any before the function is evaluated. However, the arguments are not evaluated as such irrespective of whether the function name corresponds to EXPR or MACRO in the LISP definition. It is this facility that

gives the user the power he wants out of such a system. For example knowledge for deciding whether a department is an HSS Department, one has to keep the following piece of information.

```
(HSS-DEPT PSY)
(HSS-DEPT PHI)
(HSS-DEPT SOC)
⋮
```

However, with the facility of special function one can code this knowledge simply as one rule

```
((HSS-DEPT ?X) <= ((DEPT ?X) (ONE-OF ?X (PSY PHI SOC..))))
```

Also relations like OLD YOUNG can be coded easily

```
((YOUNG ?X) <= ((AGE ?X ?Y) (LESS-THAN ?Y 30)))
```

Argument: An argument can be either a constant, a predicate-variable or a special variable.

a) Constant any character string that starts with an alphabet without embedded blanks is treated as a constant. Numbers are also treated as constants. Constants as arguments of a relation represent a realization of the relation.

For example,

```
(PRIME-MINISTER INDIA RAJIV)
```

is one realization of the function

```
(PRIME-MINISTER ?NATION ?PERSON)
```

Realizations form the data-base of the system i.e. the facts in the knowledge base.

b) **Predicate Variable:** Any identifier with ? as the first character is treated as a predicate variable. Predicate variables are assumed to be globalized i.e.

$((\text{BARKS } ?X) \leq (\text{DOG } ?X))$

is assumed to be $\forall X[\text{DOG}(X) \Rightarrow \text{BARKS}(X)]$

Predicate variables act as carriers to bring out facts from the knowledge base. When two relations are unified, a predicate variable would match with anything found as a counter-part and it is said to be bound to that counterpart. Hence unifying

$(\text{DESCENDANT } ?X \text{ RAMA})$ with

$(\text{DESCENDANT DASHARATHA RAMA})$ would result in binding

X to Dasharatha. However,

$(\text{FATHER } ?X \text{ RAMA})$ when matched with

$(\text{FATHER RAM DASHARATHA})$ would result in a failure.

In a rule, if a predicate variable is bound to a value, the binding is imposed on all occurrences of the variable in that rule.

c) **Special Variable:** Any identifier that starts with # is treated as a special variable. This is useful for some applications where special matching rules are required. For example, in a data-base extraction, a special matching may be used for retrieving names. One such method may be to knock-off the vowels and match only the consonents. Another possible application is the wild character search. A matching procedure for

this purpose may be defined by the user. However, consonent-matching is provided as a default definition of special matching. Thus we observe that a rule antecedent is a conjunct by default. But a consequent is always a simple function. This is necessary for backward chaining where the consequent becomes the goal. We also observe that the antecedent may contain any LISP function. Normally this facility is useful for testing the validity of some conditions. At the same time this can be used for other purposes - say for flashing messages or putting some flag. This gives the user the power to control the execution path. Also this can be cleverly used for debugging or tracing the flow of execution. Suppose user has a feeling that for a particular set-up, the inference engine should not be using a rule. So he wants to see what is happening at that stage. This can be easily accomplished by inserting a new rule with the same consequent but with a special function as antecedent. For example,

```
(VERY-SPECIAL-CASE ?X) <= (CLARIFY))
```

and CLARIFY may be defined as a simple LISP function that just calls the explanation system at this stage. [LISP interface is available to the user in IME through the command LISP]. Now the user can see ~~kw~~ what all has been done so far. After that control will be returned to the inference engine. Clause will succeed or fail depending on the value returned by the function.

4.2 INDEXING:

Normally knowledge bases are big, so that any attempt to try all the rules for a purpose is going to be thoroughly inefficient. This calls for organizing the knowledge into "chunks" so that only relevant knowledge is applied for a particular task. IME achieves this by indexing the rules and facts according to the function names. In fact all the rules for which a particular function is consequent are grouped together and stored as a property of the function name under the property RULES. Facts and dynamic data (New facts inferred by IME) are stored similarly under the properties of FACTS and DATA respectively. However, if knowledge is requested with a predicate variable sitting in the place of function name, the knowledge base manager thinks that all the knowledge is asked for a blind search and hence the entire knowledge base is returned.

4.3 INFERENCE ENGINE:

This is the heart of IME. Rules are invoked in a simple backward chaining fashion that produces an exhaustive depth-first search of an and-goal-tree. Suppose that the program is attempting to find out a solution to (JUNIOR X). All the relevant knowledge are supplied by the Knowledge Base Manager (KBM). In the absence of any directly matching facts (i.e. patterns like (JUNIOR SHAMU)) subgoals are formed by

ORIGINAL
87473

applying the first rule in the rule list. Out of these subgoals first one is taken as the current goal and the search continues, but not before completing the book keeping i.e. storing information about what to do next when the control is given to this goal again. Information is stored in a node and the tree keeps growing until some subgoal returns a definite answer. If the goal fails decisively, then the node is removed from the tree and the next alternative left for the higher goal is tried (Typically next rule in the queue is tried). Instead, if a goal succeeds, i.e. it finds some matching data, the answer is returned. At the same time information is preserved. Because in IME, indeterminate case is assumed, i.e. each goal may produce more than one answer. If the bindings found for one clause in a conjunct are such that the next conjunct fails to come up with results, then the first one is called again to give alternate solution. This requires retaining the node even after it returns an answer successfully. Like wise, the whole tree is preserved even after the first goal is achieved, because the user may ask for alternate solutions.

This results in a depth first search, and the sprouted tree is an and-tree. However, this normal search may be affected by the presence of "cyclic traps" in the rule. IME puts no restriction on recursion of rules and as a result sometimes the normal search may lead to infinite tree growth.

For example a rule of the form

$$((\text{COUPLE } ?X ?Y) \Leftarrow (\text{COUPLE } ?Y ?X))$$

is perfectly meaningful as far as the user is concerned. But suppose $(\text{COUPLE } ?X ?Y)$ is the first goal and no data is available about COUPLE. Then the subgoal is formed, which again is of the same form, but with different predicate variables. This will again call for a subgoal and the tree grows infinitely. Hence such traps are to be detected before any node goes about setting subgoals. This is achieved by checking for matching goals of higher ups. If a trap is detected, the node is wound up returning failure to the higher up.

A straight forward search is not efficient because the experience of trying to achieve a goal is not utilised for other purposes. Thus the program may go about searching deep, though a similar search had failed earlier. Or it may go about forming subgoals after subgoals to find answer to a goal, for which an answer might have been found out earlier with long search. These factors call for an "information centre" that would store the useful information out of a search and cater to the needs of others. This task is assigned to knowledge base manager. KBM watches the search closely and any useful information is stored which is then supplied to the concerned calls for knowledge. For example if KBM observes that a goal $(\text{BROTHER RAMA SEETHA})$ was unsuccessful, a note is written that $(\text{BROTHER RAMA SEETHA})$ is not achievable. Hence any

subsequent request for knowledge for achieving (BROTHER RAMA SEETHA) will not get any knowledge from KBM. Similarly, if the goal (BROTHER RAMA ?WHO) gets an answer as (?WHO = LAXMANA), KBM understands that (BROTHER RAMA LAXMANA) is true, and, if it is not already there in the data base this will be stored in dynamic data base.

But the problem is that in a search like this there may be a large number of intermediate results, which may not be worth keeping. So KBM has to decide about the worth of a piece of information. A simple criteria of checking for predicate variables is used for this purpose. An information is treated as "useful" if it does not contain any predicate variable.

Even this system is not free of problems. What if a goal was not achieved because of some cyclic traps in one of its descendants and the "cyclic trap" was actually caused because of a goal of one of his higher ups. In such cases KBM should not conclude that the goal is not achievable. This problem is solved by some more book keeping whenever a cyclic trap occurs.

Working of the inference engine may be summarized in the following algorithmic representation.


```

Form the root-node
Make this node the current-node
LOOP until task-over
  IF Leaf-node
  THEN
    IF no-cycling-trap, THEN solve the task FI.
    IF success or failure
    THEN
      Call the attention of KBM
      Report answer to higher up
      Make higher up the current-node
    ELSE ; i.e. subgoals are formed
      Form a leaf as a descendant
      Supply it with required knowledge
      Make new leaf the current-node
    FI
  ELSE,; i.e. if the current node is not a leaf
    IF call is from higher up ; that means the call is
      to supply alternate answers.
    THEN
      delete the answer received by the last descendant
      Make last descendant the current node
    ELSE ; that means a descendant has returned some answer
      IF Success reported
      THEN
        Store the answer
        IF subgoals exist
        THEN
          form a new leaf and assign the-
          -next goal with relevant knowledge
          make that leaf current node
        ELSE report consolidated answer to-
          -higher up.
          call the attention of KBM
        FI
      ELSE ; a descendant has returned with failure
        remove the leaf from the tree.
        IF some descendants still exist
        THEN
          call the latest descendant after-
          -deleting the report.
        ELSE ; the node has now become a
          ; leaf as all the descendants
          ; are deleted from the tree
        FI
      FI
    FI
  FI
FI
END LOOP

```

Following informations are stored for a node:

1. GOAL
2. CALLED-BY-NODE
3. CURRENT-KNOWLEDGE-STATUS : (This is an information regarding the portion of the relevant knowledge that is not tried so far)
4. ANSWER : (Bindings of the unification of goal with the consequent of the rule used.)
5. DESCENDANTS : (List of descendants in the order of their creation)
6. REPORTS : (Answers returned by descendants)
7. SUBGOALS : (Subgoals to be tried)
8. CYCLIC FLAG : (Indicates whether a cyclic trap had occurred between a higher up and a descendant)

Let us examine an example. Let the tree be as shown in Fig.(4.2) at some stage of a search for a YOUNG SENATOR. The task would be represented internally as

```
((YOUNG ?WHO)(SENATOR ?WHO))
```

So the root node will have two subgoals and the first one will be tried first which will lead to subgoals

```
((PERSON ?WHO) ( ?WHO AGE ?AGE) (LESS THAN ?AGE 30))
```

Using the rule

```
((YOUNG ?X) <= ((PERSON ?X)(?X AGE ?AGE)
                  (LESSER-THAN ?AGE 30)))
```

and so on.

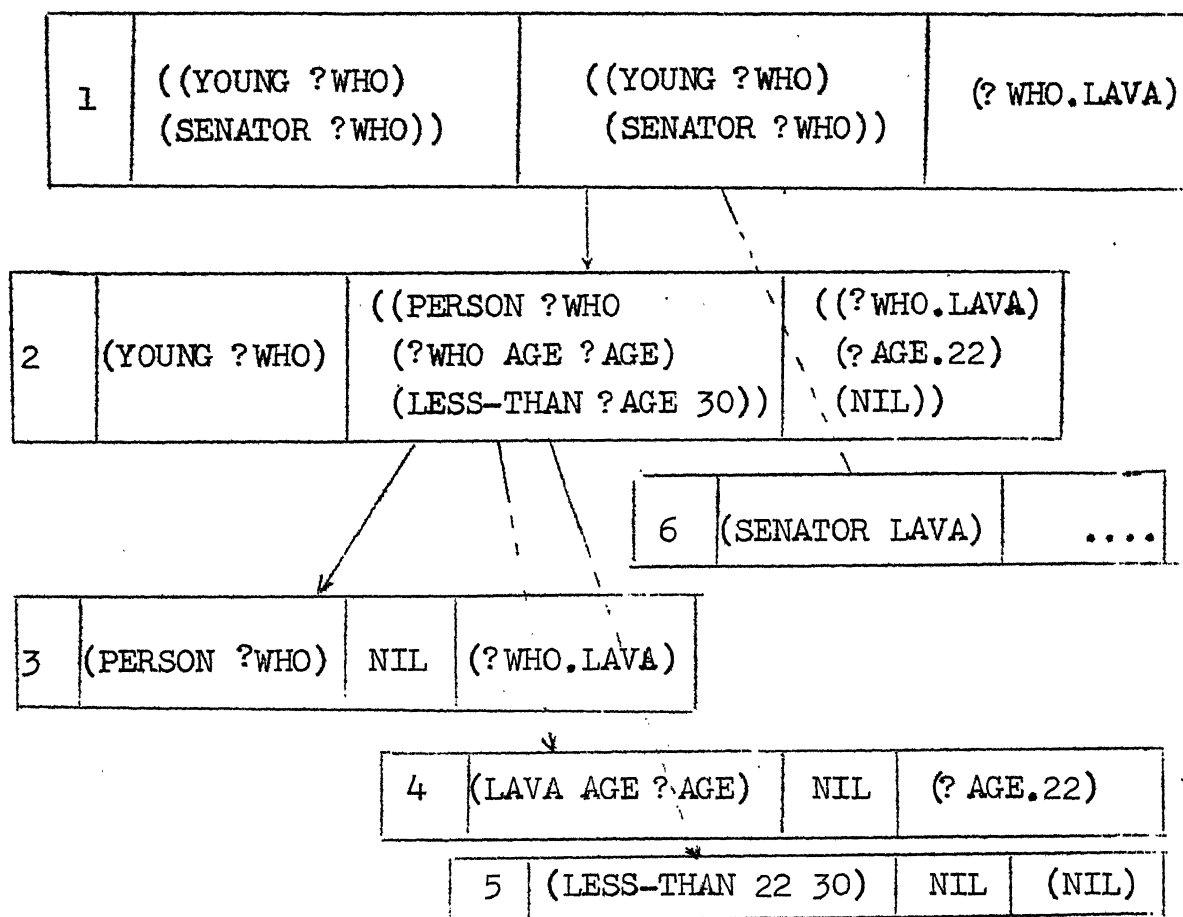


FIG. 4.2(A)

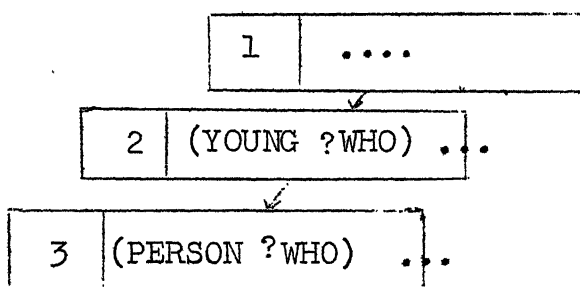


FIG. 4.2(B).

At a stage shown in the figure we study the contents of the nodes 1 and 3 which may give an insight into the working of the program.

NODE 1 :

DESCENDANT-OF : NIL ; Indicates that this is the root node.

GOAL : ((YOUNG ?WHO) (SENATOR ?WHO))

; Only root node can have conjunct goals
; in which case they are directly taken
; as subgoals.

ANSWER : NIL ; no matching was done for the goals.

KNOWLEDGE : NIL ; No knowledge is taken, as subgoals are
directly formed.

SUBGOALS : ((YOUNG ?WHO) (SENATOR ?WHO)).

DESCENDANTS: (2 6) ; It has two direct descendants.

REPORTS : ((?WHO.LAVA)); there is only one report which means
that the report from the other is awaited.

Take node 3 which is a leaf.

NODE 3 :

DESCENDANT OF : 2

GOAL : (PERSON ?WHO)

ANSWER : (?WHO. LAVA)

KNOWLEDGE : ((PERSON LAVA) (PERSON RAMA)...))

SUBGOALS : NIL

DESCENDANTS : NIL

REPORTS : NIL

Now, should node 6 fail, it will be deleted and 1 would ask 2 to come up with alternate answer and 2 inturn sounds 5. When 5 is unable to come up with alternate answer, 5 is deleted and 4 is sounded who again fails as age can be unique (according to the knowledge available to 4) and so 3 is called into service again. At this stage the tree would be as shown. Now 3 would come up with the alternate solution (WHO.RAMA) and the process continues.

4.4 EXPLANATION SYSTEM:

Questions are thrown to the user whenever inference engine encounters with a function that is askable. However, the user can ask for reasons, at which stage explanation system is called. A trace would highlight the power of explanation system.

What is your age

?WHY

If I prove

[1-0] Your age is AGE

[1-1] (LESS-THAN ?AGE 28)

Then I can conclude that

You are young.

~~#~~ WHY

I have found out that

[2-0] Your qualification is MTECH

[2-1] You have LOT-OF money

[2-2] You are ready to spend 5 years on education

If I prove

[2-3] You are YOUNG

Then I can conclude that

You are advised to take up PHD.

~~#~~ HOW [2-1]

I have found out that

[3-0] You are ready to spend Rs. 10000

[3-1] (GREATER-THAN 10000 5000)

So I concluded

You have LOT-OF money

~~#~~ OK

As can be seen from the examples, IME's explanation is just a tree traversal. Control will move up or down the tree depending on user's requirement. However, instead of dumping the information as it is stored internally, IME utilises the simple technique of slot filling to produce English-like explanation. Translations are stored for each function (Default is to print as a LESP list as it is stored internally). Translation is printed, with ~~#~~ N replaced by N-th argument of the function in TRANS property.

However explanation system does not understand English sentences. It's vocabulary is limited to very few words like HOW, WHY, OK etc.

4.5 IME TOP-LEVEL:

IME has a parser with limited lexicon. Commands are usually rigid. Facts and rules can be added or deleted. Direct access to LISP is also provided, with the command LISP. Editing facts and rules is also possible (LISP Editor is used directly for this purpose). Current knowledge base can be viewed selectively.

A complete list of commands which IME's TOP-LEVEL recognizes is listed in Appendix A.

CHAPTER V

RESULTS AND CONCLUSIONS

5.1 SUMMARY:

In this thesis we have discussed about IME - a tool for developing knowledge based systems. Attempt was made to distinguish clearly between the domain dependent expert knowledge from the domain independent control structures. The aim was to develop a skeletal system using which, expert systems can be built readily.

The system is designed accordingly. A clear distinction is drawn between domain knowledge and general purpose problem solving capabilities. Inference engine is designed to cover the needs of a variety of situations. Switches are provided to inform the inference engine about the specific needs of the problem. Inference engine is equipped to take care of self referencing rules, and to produce all possible solutions. A depth first tree search technique is adopted for deduction while state saving techniques are used for back-tracking. The inference tree is maintained to facilitate explanation and generation of alternate solutions. Intermediate results are made use of to reduce the search. A proper book keeping procedure helps in pruning the paths that will not produce any results.

Explanation system produces convincing explanations and justifications for the conclusions as well as the questions asked.

Rules and facts can be added or deleted interactively. IME imposes no restriction on rules. Only a simple syntactical checking is made before accepting a new rule.

Online help is available about the acceptable commands.

5.2 LIMITATIONS:

IME's inference engine is simple but not efficient mainly because of two factors.

1. When more than one rule are applicable, no effort is made to select the most prospective rule. The selection is sequential. In simple applications this may not be a serious problem as the rules can be ordered properly in the knowledge-base. But if the relative importance depends on the situation (which is true in most of the situations) then dynamic re-ordering has to be done. IME does provide a provision to take care of this but it is a poor consolation. What IME gives is just an idea but no clues are given as such.
2. While testing the conjuncts in the antecedant of a rule IME does not look ahead. It blindly goes about testing them in the same order. A one step look ahead might help a lot especially in situations where number of conjuncts in a rule are more than 3 on an average [7].

IME may be upgraded with techniques for re-ordering rules and to have a quick look at all the conjuncts in a rule before trying to test one at a time.

IME accepts rules and facts only as syntactically correct LISP lists. No natural language capability is exhibited while accepting inputs from user. IME may be taught about a few types of sentences and key words may be taken as inputs while developing a particular application system.

REFERENCES

1. Basden, Andrew., ON THE APPLICATION OF EXPERT SYSTEMS, Int. J. Man-Machine Studies (1983) 19, 461-477.
2. Charniack, S.J. et.al., ARTIFICIAL INTELLIGENCE PROGRAMMING, Addison-Wesley Pub.Co.
3. Dahl Veronica, LOGIC PROGRAMMING AS A REPRESENTATION OF KNOWLEDGE, IEEE Computer, Oct. 83.
4. Davis, Randall., D.B. Lenat., KNOWLEDGE-BASED SYSTEMS IN ARTIFICIAL INTELLIGENCE, McGraw-Hill Int. Book Company, 1982.
5. Hayes-Roth, Frederick., et.al., BUILDING EXPERT SYSTEMS, Addison-Wesley Publishing Company, 1983.
6. Hayes-Roth, Frederick., THE KNOWLEDGE BASED EXPERT SYSTEM: A TUTORIAL, IEEE Computer, Sept. 1984.
7. Mittal, Sanjay., et.al., PATREC: A KNOWLEDGE-DIRECTED DATA-BASE FOR A DIAGNOSTIC EXPERT SYSTEM, IEEE Computer, Sept. 1984.
8. Michie, Donald., EXPERT SYSTEMS IN THE MICRO ELECTRONIC AGE, Edinburgh Univ. Press, 1979.
9. Mylopoulos, T.S., and John K. Tsotsos., BUILDING KNOWLEDGE-BASED SYSTEMS: THE PSN EXPERIENCE, IEEE Computer, Oct. 1983.
10. Nilsson, N.J., PRINCIPLES OF ARTIFICIAL INTELLIGENCE, Springer-Verlag, 1981.
11. Shortliffe, E.H., MYCIN: COMPUTER BASED MEDICAL CONSULTATION SYSTEM, American Elsevier, New York, 1976.
12. Wiederhold, G.O., KNOWLEDGE AND DATABASE MANAGEMENT, IEEE Trans. Software Eng. June, 1984.

APPENDIX A

TOP LEVEL COMMANDS RECOGNIZED BY IME

- HELP** Prints a brief help message about the system.
- <assertion>**. This is to add a fact to the knowledge base
e.g. (BROTHER RAMA LAXMANA). Note the period
at the end. It is a must.
- <Request>** This is the command for asking IME to solve some
problem. Please note ? at the end. This is a
must.
e.g. ((YOUNG ?WHO) (SENATOR ?WHO))
- <rule>** (<consequent> <= <antecedent>)
This adds the rule to the knowledge-base.
e.g. ((FOO ?X ?Y) <= ((FIE ?Y ?Z) (FAZ ?X ?Z)))
- LIST RULES ID** This would list all the rules known to IME
about ID.
e.g. LIST RULES FOO Would list all the rules
known to IME about FOO.
- LIST FACTS ID** Similar to the above but prints the facts known
to IME.
- DELETE RULE ID** It first prints all the rules currently known to
IME and asks for the sl.no. of the rule to be

deleted. If a number not in the range
is given as response the command will be skipped.

DELETE FACT ID Similar to the above ~~m~~command.

EDIT ID This is useful only if the user knows about
the LISP Editor.

LISP <expr> Useful for calling LISP functions.
<expr> is evaluated and printed.

BYE, QUIT, STOP, END END the session

CYCLING To indicate to IME that cycling or infinite
looping may occur in the application.

NO-CYCLING Asking IME not to worry about "cycling traps".

RESULTS To advise IME to store intermediate results and
use as facts.

NO-RESULTS Advising IME not to bother about making use of
intermediate results.

BLIND-ALLEY Advising IME to keep track of useless paths.

NO-BLIND-ALLEY Not to worry about the above facility.

TRANS ID (Translation)

TRANS stores the translation as the property of
ID and is used for English translation. Trans-
lation should contain slots for putting the
arguments if any

e.g. TRANS BROTHER (~~#~~ 2 is brother of ~~#~~1)
would mean that

(BROTHER RAMA LAXMANA) will be translated as
LAXMANA is brother of RAMA.

If TRANS property is not specified, internal representation is printed as LISP list.

PROMPT ID Prompt

This tells IME that **prompt** should be printed to ask questions about ID .

Note that the prompt text should be enclosed in double quotes.

NOT-ASKABLE ID This undoes the effect of PROMPT

SAVE SAVE saves the knowledge base and other details of the system on to the disk so that they can be used for subsequent sessions.